Efficient Conjugate Gradient Algorithms for Computation of the Manipulator Forward Dynamics

Amir Fijany and **Robert E. Scheid**

Jet Propulsion Laboratory, California Institute of Technology
Pasadena, California

Abstract

In this paper, we investigate the applicability of conjugate gradient algorithms for computation of the manipulator forward dynamics. The redundancies in the previously proposed conjugate gradient algorithm is analyzed [7]. A new version is developed which, by avoiding these redundancies, achieves a significantly greater efficiency. A preconditioned conjugate gradient algorithm is also presented. A diagonal matrix whose elements are the diagonal elements of the inertia matrix is proposed as the preconditioner. In order to increase the computational efficiency, an algorithm is developed which exploits the synergism between the computation of the diagonal elements of the inertia matrix and that required by the conjugate gradient algorithm.

I. INTRODUCTION

The manipulator forward dynamics problem, which concerns the determination of the motion resulting from the application of a set of joint forces/torques, is essential for the dynamic simulation of robot manipulators. The motivation for devising fast algorithms for the forward dynamics solution stems from applications which require extensive off-line simulation as well as applications which require real-time dynamic simulation. In particular, for many anticipated space teleoperation applications, a faster-than-real-time simulation capability will be essential. In fact, in the presence of the unavoidable delay in information transfer, such a capability would allow a human operator to preview a number of scenarios before run-time [1].

The forward dynamics problem can be stated as follows: given the vector of the actual joint positions (Q) and velocities (Q), and the vector of applied joint forces/torques (τ), find the vector of the joint accelerations ($\ddot{Q}$). Integrating $\ddot{Q}$ leads to the new values for Q and $\dot{Q}$. The process is then repeated for the next τ. The first step in the computation of the forward dynamics is to derive a linear relation (for the given Q) between the vector of joint accelerations and the vector of joint inertia forces/torques. Given the dynamic equations of motion as

$$A(Q)\ddot{Q} + C(Q,\dot{Q}) + G(Q) + J^t(Q)F_E = \tau \tag{1}$$

and the bias vector (b) as

$$b = C(Q,\dot{Q}) + G(Q) + J^t(Q)F_E \tag{2}$$

the linear relation is derived as

$$A(Q)\ddot{Q} = \tau - b = \Gamma \tag{3}$$

where A(Q) is an nxn symmetric, positive definite, inertia matrix and J is

the 6xn Jacobian matrix (t denote matrix transpose). Q, $\dot{Q}$, $\ddot{Q}$, τ, b, $\Gamma \in R^n$, and F_E is the 6x1 vector which is a compact representation of the external force (f_E) and moment (n_E) exerted on the End-Effector (EE). The bias vector represents the contribution due to the nonlinear terms as well as the external force and moment. Hence, Γ stands for the vector of applied inertia forces/torques. The bias vector can be obtained by computing the inverse dynamics, using the Newton-Euler (N-E) formulation [2], for the actual value of Q, $\dot{Q}$, and F_E while setting $\ddot{Q}$ to zero. The evaluation of b and Γ, i.e., the derivation of Eq. (3), is necessarily the first step in the computation of forward dynamics.

The proposed algorithms for computation of the forward dynamics differ in their approaches to solving Eq. (3), which directly affect their asymptotic computational complexity. These algorithms can be classified as $O(n)$ algorithms [3]-[6], the $O(n^2)$ algorithms [7], and the $O(n^3)$ algorithms [7]. However, any analysis of the efficiency of these algorithms should be based on the realistic size of the problem, i.e. the number of Degrees-Of-Freedom (DOF). In fact, the comparative study in [3] shows that the $O(n^3)$ composite rigid-body algorithm is the most efficient for n less than 12. It also shows that, due to the large coefficient of n^2 terms on the polynomial complexity, the conjugate gradient algorithm of [7] does not become more efficient than the composite rigid-body algorithm except for very large n, making the algorithm almost impractical.

In this paper, we develop two conjugate gradient algorithms which are significantly more efficient than that of [7]. The better efficiency of these algorithms is mainly achieved by a significant reduction of the coefficient of n^2 terms on the polynomial complexity. The first is a Classical Conjugate Gradient (CCG) algorithm which improves the computation cost of each iteration by eliminating the redundancy in the *extrinsic* equations, i.e., by a better choice of coordinate frame for projection of the *intrinsic* equations. With this reduction in the cost of each iteration, a further efficiency can be achieved by reducing the number of iterations through the use of a preconditioner. The second is a Preconditioned Conjugate Gradient (PCG) algorithm which uses a positive definite diagonal matrix, whose elements are the diagonal elements of the inertia matrix, as a preconditioner. An efficient algorithm for computation of the diagonal elements of the inertia matrix is also developed.

However, despite these improvements, the developed algorithms are, in general, still less efficient than the best $O(n^3)$ algorithm. It should be pointed out that the efficiency of this algorithm is further increased by a recently developed algorithm [8]-[9] which achieves greater efficiency in computing the inertia matrix over the composite rigid-body algorithm in [7]. Despite the improvement in the efficiency of the serial algorithms, even the fastest serial algorithm is far from providing the required efficiency for real-time or faster-than-real-time simulation. This observation clearly suggests that the exploitation of parallelism in the computation is the key factor in achieving the desired efficiency.

The analysis of the parallel efficiency of different algorithms is more complex than that of the serial efficiency [9]. Our investigation indicates that the PCG algorithm presents excellent features for parallel computation [10]. In fact, the parallel version of the PCG algorithm, while requiring a simple architecture, may potentially become the most efficient alternative for parallel computation of the forward dynamics. In fact, such a potential has motivated us to further investigate the PCG algorithm and the impact of the preconditioning on its convergence. In this paper the preliminary results of our investigation are presented.

This paper is organized as follows. In Section II, the CCG and PCG algorithms are briefly reviewed and the particular features of these algorithms in the context of the forward dynamics computation are discussed. In Section III, the CCG algorithm is developed. In Section IV, the PCG algorithm and the algorithm for computation of the diagonal elements of the inertia matrix are presented. Finally, some discussion and concluding remarks are made in Section V.

II. CONJUGATE GRADIENT METHOD AND RESULTING ALGORITHMS

The conjugate gradient method is one of the most widely used methods for the iterative solution of linear systems of equations such as

$$Ax = b \qquad x,\ b \in \mathbb{R}^n \tag{4}$$

where $A \in \mathbb{R}^{nxn}$ is a symmetric positive-definite matrix. An attractive feature of the method is the guarantee of the convergence in at most n steps. Several developments have contributed to the wide application of the method [13]; they include analysis and experimentation leading to the identification of the most stable versions of the method, an understanding of its error propagation, and the fact that the solution of Eq. (4) arises in many applications.

The discussion given here is mainly based on the treatment found in [12] where the basic algorithm is given as follows:

$x_0 = 0$

$r_0 = 0$

For $j = 1, 2, \ldots, n$

if $r_{j-1} = 0$ then set $x = x_{j-1}$ and quit

else

$$\beta_j = r_{j-1}^t r_{j-1} / r_{j-2}^t r_{j-2} \qquad \beta_1 \equiv 0 \tag{5}$$

$$p_j = r_{j-1} + \beta_j p_{j-1} \qquad p_1 \equiv 0 \tag{6}$$

$$\alpha_j = r_{j-1}^t r_{j-1} / p_j^t A p_j \tag{7}$$

$$x_j = x_{j-1} + \alpha_j p_j \tag{8}$$

$$r_j = r_{j-1} - \alpha_j A p_j \tag{9}$$

$x = x_n$

This is the Classical Conjugate Gradient (CCG) algorithm which has been analyzed in considerable detail under general conditions.

The interest in the conjugate gradient method has been further increased by

the development of the preconditioning strategies to accelerate convergence of the algorithm. Furthermore, while the CCG algorithm and its preconditioned versions are not naturally suitable for parallel computation, they are well matched for vector supercomputers, i.e., they can be efficiently vectorized [13]-[14]. The key concept in achieving a faster convergence resides in improving the condition of matrix A by preconditioning [12]. Let C be some nonsingular symmetric matrix and define $\tilde{A} = C^{-1}AC^{-1}$, $\tilde{b} = C^{-1}b$, and $\tilde{x} = C^{-1}x$. Then the algorithm can be applied to the equivalent transformed system $\tilde{A}\,\tilde{x} = \tilde{b}$ where for an appropriate choice of C the convergence may be accelerated considerably. Let $M = C^2$. The algorithm (for n steps) is written as [12]:

$x_0 = 0$

$r_0 = 0$

For $j = 1, 2, \ldots, n$

if $r_{j-1} = 0$ then set $x = x_{j-1}$ and quit

else

Solve $MZ_{j-1} = r_{j-1}$ for Z_{j-1} (10)

$\beta_j = Z_{j-1}^t r_{j-1} / Z_{j-2}^t r_{j-2}$ $\qquad \beta_1 \equiv 0$ (11)

$p_j = Z_{j-1} + \beta_j p_{j-1}$ $\qquad p_1 \equiv 0$ (12)

$\alpha_j = Z_{j-1}^t r_{j-1} / p_j^t Ap_j$ (13)

$x_j = x_{j-1} + \alpha_j p_j$ (14)

$r_j = r_{j-1} - \alpha_j Ap_j$ (15)

$x = x_n$

This is the Preconditioned Conjugate Gradient (PCG) algorithm and the symmetric positive definite matrix M is called the preconditioner. In order for M to be effective as a preconditioner, it is essential to be able to easily solve the linear systems in Eq. (10). A well chosen preconditioner can lead to rapid convergence, often after $O(n^{1/2})$ iterations [12]. Note that if $M^{-1} = A^{-1}$, then the iteration converges immediately. So one hopes that when $M^{-1} \approx A^{-1}$ (in some sense) the iteration converges very quickly. In fact, this is what has been shown in [15]. As a result, if the matrix A is diagonally dominant then M = Diag (A) may be an excellent preconditioner since M closely approximates A. Furthermore, with the a diagonal matrix the solution of Eq. (10) is trivial. The choice of M = Diag (A) is known as Diagonal Scaling or PCG-DS. Note that, compared to the cost of each iteration of CCG, such a choice leads to only an additional cost of n divisions per iteration of PCG-DS. Given the faster convergence, this represents an efficient tradeoff which explains the preference for the use of PCG-DS over CCG even where A is not diagonally dominant.

However, the serial and parallel computation of the conjugate gradient algorithms, when applied to the forward dynamics solution, differs from its application to more generic problems. In fact, it is usually assumed that the matrix A is given which is not the case for the forward dynamics problem.

For serial processing, note that, the basic operation in the CCG and PCG algorithms is the matrix-vector multiplication in Eqs. (7) and (13) with the computation complexity of $O(n^2)$. Given n iterations, this leads to $O(n^3)$ computational complexity of the algorithms. For forward dynamics problem, this operation represents the evaluation of the vector of joint inertia forces/torques, i.e., $\Gamma(j)$, for a given vector of joint acceleration (p_j), which can be computed in $O(n)$ steps, using the N-E formulation. This can be done for CCG algorithm without explicit computation of A which has also been exploited in [7]. Note that, the derivation of the dynamic models of the industrial manipulators, in symbolic form, shows that their inertia matrices can be practically considered as diagonal dominant [16]. Therefore, the PCG-DS algorithm can be expected to achieve a rapid convergence in solving the forward dynamics problem. However, the application of PCG-DS algorithm requires the computation of the diagonal elements of A. Hence, the algorithmic efficiency in computing the diagonal elements is a key factor in the successful application of PCG-DS algorithm to the forward dynamics solution.

In the context of the forward dynamics solution, the CCG and PCG-DS also provide suitable features for parallel processing. Exploiting maximum parallelism, the matrix-vector multiplication in Eqs. (7) and (13) can be performed in $O(\log_2 n)$ steps with $O(n^2)$ processors. However, besides using too many processors, exploitation of maximum parallelism requires a complex processor interconnection. For the forward dynamics problem, this operation, as is shown in [18], can be performed in $O(\log_2 n)$ steps with n processor and a rather simple interconnection. This leads to the $O(n\log_2 n)$ parallel CCG algorithm. It is shown that, using the same architecture as in [18], the diagonal elements of the inertia matrix can be computed in $O(\log_2 n)$ steps [11]. This implies that, if PCG-DS algorithm converges in $O(n^{1/2})$ iterations, then its parallel version can achieve a computational time of $O(n^{1/2}\log_2 n)$ with n processor and a simple processor interconnection structure. In fact, the parallel PCG-DS may represent the fastest stable algorithm for computation of the forward dynamics problem [10].

III. THE CCG ALGORITHM

III.1 Notations and Preliminaries

The N-E formulation can be expressed as a function g_1 which, given Q, $\dot{Q}$, $\ddot{Q}$, and F_E, evaluates τ as [4]:

$$\tau = g_1(Q, \dot{Q}, \ddot{Q}, F_E) \tag{16}$$

The matrix-vector operation in Eqs. (7) and (13) is a special application of g_1 which evaluates a set of vectors of inertia forces/torques as:

$$\Gamma(j) = g_1(Q_a, 0, \ddot{Q}_j, 0) = g_2(Q_a, \ddot{Q}_j) \tag{17}$$

where Q_a is the vector of joint positions representing the manipulator's configurations for which Eq. (17) is evaluated for a set of $\ddot{Q}_j$'s.

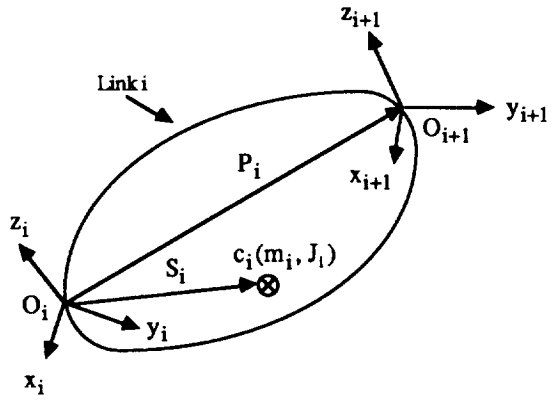

Fig.1. Link, Frames, and Kinematic and Dynamic Parameters

$q_i, \dot{q}_i, \ddot{q}_i$	position, velocity, and acceleration of joint i, respectively.
$\dot{\omega}_i$	Angular acceleration of link i
$\dot{V}_i$	Linear acceleration of link i (point O_i).
$\dot{V}_{ic}$	Linear accelerations of center of mass of link i (point cm_i).
F_i and N_i	Force and moment exerted on center of mass of link i.
f_i and n_i	Force and moment exerted on link i by link i-1.

Table I. Notion Used in the Derivation of the Algorithms.

The major redundancy in the evaluation of Eq. (17) by the algorithm of [7] results from the choice of coordinate frame for projection of the intrinsic equations. Note that the evaluation of the original N-E formulation in link coordinate frames requires $O(n)$ transformations for link-to-link propagation of the variables. Hence, using the link frames for n times evaluation of Eq. (17), as is done in [7], requires $O(n^2)$ transformations. However, if n times evaluation of Eq. (17) is performed in a fixed frame then only $O(n)$ transformations for projection of the vectors and the tensors are required.

In deriving the algorithms, we first develop the intrinsic equations, i.e., the coordinate-free representation of equations. This provides a suitable abstraction since the equations can be derived from the intrinsic physical relationships, which are independent of any coordinate frame. More important, this allows us to distinguish between the redundancy in the intrinsic and that in the extrinsic equations. In order to derive the intrinsic equations, we need to recall some notations. In this paper, according to Gibbs notation, vectors are underlined once and tensors (tensors of order 2) twice. The projection of the vectors and the tensors results in 3x1 (column matrix) and 3x3 scalar matrix wherein the superscript denotes the coordinate frame on which the projection is performed. To any vector $\underline{V}$ a tensor $\hat{\underline{\underline{V}}}$ can be associated whose projection is a 3x3 skew symmetric scalar matrix as:

$$\hat{V} = \begin{bmatrix} 0 & -V_{(z)} & V_{(y)} \\ V_{(z)} & 0 & -V_{(x)} \\ -V_{(y)} & V_{(x)} & 0 \end{bmatrix}$$

Note that $\hat{\underline{\underline{V}}}_1\underline{V}_2 = \underline{V}_1 x \underline{V}_2 = -\underline{V}_2 x \underline{V}_1 = -\hat{\underline{\underline{V}}}_2\underline{V}_1$. Also, a set of notations, presented in Fig. 1 and Table I, are used in the derivation of the algorithms.

III.2 A Variant of The N-E Formulation

Let us write the N-E formulation for link i (Fig. 1) with the nonlinear terms being excluded.

$$\dot{\underline{\omega}}_i = \dot{\underline{\omega}}_{i-1} + \underline{z}_i \ddot{q}_i \tag{18}$$

$$\dot{\underline{V}}_i = \dot{\underline{V}}_{i-1} + \dot{\underline{\omega}}_{i-1} x \underline{P}_{i-1} \tag{19}$$

$$\dot{\underline{V}}_{ic} = \dot{\underline{V}}_i + \dot{\underline{\omega}}_i x \underline{S}_i \tag{20}$$

$$\underline{F}_i = m_i \dot{\underline{V}}_{ic} \tag{21}$$

$$\underline{N}_i = \underline{\underline{J}}_i \dot{\underline{\omega}}_i \tag{22}$$

$$\underline{f}_i = \underline{F}_i + \underline{f}_{i+1} \tag{23}$$

$$\underline{n}_i = \underline{N}_i + \underline{S}_i x \underline{F}_i + \underline{n}_{i+1} + \underline{P}_i x \underline{f}_{i+1} \tag{24}$$

$$\Gamma_i = \underline{z}_i \cdot \underline{n}_i \tag{25}$$

where Γ_i is the ith component of Γ which indicates the inertia force/torque of joint i. Eqs. (18)-(25) describe the procedure for computation of the vector $\Gamma(j)$ or the function g_2. Note that, for the sake of simplicity, an all revolute joints manipulator is considered. However, with small changes, the results can be extended to the manipulator with sliding joint(s).

A variant of g_2 can be derived by replacing Eqs. (20)-(22) into Eqs. (23) and (24) as

$$\underline{f}_i = m_i \dot{\underline{V}}_i + \dot{\underline{\omega}}_i x (m_i \underline{S}_i) + \underline{f}_{i+1} \tag{26}$$

$$\begin{aligned} \underline{n}_i &= \underline{\underline{J}}_i \dot{\underline{\omega}}_i + \underline{S}_i x [m_i \dot{\underline{V}}_i + \dot{\underline{\omega}}_i x (m_i \underline{S}_i)] + \underline{n}_{i+1} + \underline{P}_i x \underline{f}_{i+1} \\ &= (\underline{\underline{J}}_i - m_i \hat{\underline{\underline{S}}}_i \hat{\underline{\underline{S}}}_i) \dot{\underline{\omega}}_i + (m_i \underline{S}_i) x \dot{\underline{V}}_i + \underline{n}_{i+1} + \underline{P}_i x \underline{f}_{i+1} \end{aligned} \tag{27}$$

The terms $\underline{\underline{J}}_i - m_i \hat{\underline{\underline{S}}}_i \hat{\underline{\underline{S}}}_i$ and $m_i \underline{S}_i$ represent the first and the second moment of mass of link i with respect to point O_i which are designated as $\underline{\underline{k}}_i$ and $\underline{h}_i$, respectively. Note that $\underline{\underline{k}}_i$ and $\underline{h}_i$ are constant in link i coordinate frame, i.e., coordinate frame i+1, and can be given as the link parameters. The variant of the N-E formulation for computation of the vector Γ, designated as g_3, is written as:

$$\dot{\underline{\omega}}_i = \dot{\underline{\omega}}_{i-1} + \underline{z}_i \ddot{q}_i \tag{28}$$

$$\dot{\underline{V}}_i = \dot{\underline{V}}_{i-1} + \dot{\underline{\omega}}_{i-1} x \underline{P}_{i-1} \tag{29}$$

$$\underline{f}_i = m_i \dot{\underline{V}}_i + \dot{\underline{\omega}}_i x \underline{h}_i + \underline{f}_{i+1} \tag{30}$$

$$\underline{n}_i = \underline{\underline{k}}_i \dot{\underline{\omega}}_i + \underline{h}_i x \dot{\underline{V}}_i + \underline{n}_{i+1} + \underline{P}_i x \underline{f}_{i+1} \tag{31}$$

$$\Gamma_i = \underline{z}_i \cdot \underline{n}_i \tag{32}$$

In the above procedure the explicit computation of the linear acceleration of, and the force and the moment exerted on, the link's center of mass is avoided. Note that the computation cost of g_2 and g_3 is the same. However,

if the equations of both procedures are projected on some fixed coordinate frame then evaluation of g_2 requires the transformation of J_i and S_i while that of g_3 requires the transformation of k_i and h_i. For the CCG algorithm, since the evaluation of the original N-E formulation, i.e., g_1, requires the transformation of J_i and S_i, it is more efficient to use g_2. However, g_3 will be used to derive the algorithm for computation of the diagonal elements of the inertia matrix and the evaluation of Γ in PCG-DS algorithm.

III.3 Computation of the CCG Algorithm

As stated before, it is more efficient to project the equations on some fixed frame. To do so, $\underline{P}_i$, $\underline{S}_i$, $\underline{z}_i$, and $\underline{\underline{J}}_i$ should be projected onto the fixed frame. We use the EE (n+1 th) coordinate frame which is slightly more efficient since $\underline{P}_n$, $\underline{S}_n$, $\underline{z}_n$, and $\underline{\underline{J}}_n$ are constant in this coordinate frame. Let m and a denote the cost of multiplication and addition, respectively. The computational steps of the CCG algorithm are performed as follows where, for each step, its computational cost is also indicated.

Step 1: Projection of the vectors and the tensors

For i = 1, 2, ..., n

1) Evaluate ${}^{i+1}R_i$

2) $${}^{n+1}R_i = {}^{n+1}R_{i+1}\,{}^{i+1}R_i \qquad (33)$$

3) $${}^{n+1}Z_i = {}^{n+1}R_i z_0 \qquad \text{With } z_0 = [0 \;\; 0 \;\; 1]^t \qquad (34)$$

4) $${}^{n+1}S_i = {}^{n+1}R_i\,{}^{i}S_i \qquad (35)$$

5) $${}^{n+1}P_i = {}^{n+1}R_i\,{}^{i}P_i \qquad (36)$$

6) $${}^{n+1}J_i = {}^{n+1}R_i\,{}^{i}J_i\,{}^{i}R_{n+1} \qquad (37)$$

The computation cost of this step is obtained as $4nm + (n-1)(96m + 63a)$. In the following the absence of the superscript denotes that the vectors and the tensors are described with respect to the EE coordinate frame.

Step 2: Computation of $r_0 = \tau - \tau_a$

1) Compute $\tau_a = g_1(Q_a, \dot{Q}_a, \ddot{Q}_a, F_e)$

a) For i = 1, 2, ..., n

$$\omega_i = \omega_{i-1} + z_i \dot{q}_{ai} \qquad \omega_1 = 0 \qquad (38)$$

$$\dot{\omega}_i = \dot{\omega}_{i-1} + \omega_{i-1} \times z_i \dot{q}_{ai} + Z_i \ddot{q}_{ai} \qquad \dot{\omega}_1 = 0 \qquad (39)$$

$$\dot{V}_i = \dot{V}_{i-1} + \dot{\omega}_{i-1} \times P_{i-1} + \omega_{i-1} \times (\omega_{i-1} \times P_{i-1}) \qquad \dot{V}_1 = GZ_1 \qquad (40)$$

$$\dot{V}_{ic} = \dot{V}_i + \dot{\omega}_i \times S_i + \omega_i \times (\omega_i \times S_i) \qquad (41)$$

$$F_i = m_i \dot{V}_{ic} \qquad (42)$$

$$N_i = J_i \dot{\omega}_i + \omega_i \times (J_i \omega_i) \qquad (43)$$

b) For i = n, n-1, ..., 1

$$f_i = F_i + f_{i+1} \qquad f_{n+1} = f_E \qquad (44)$$

Algorithm	General		n = 6	
	Mul.	Add.	Mul.	Add.
CCG in [7]	$76n^2+120n-21$	$56n^2+87n-6$	3435	2532
This paper	$47n^2+177n-117$	$46n^2+118n-87$	2637	2277

Table II. Comparison of the CCG algorithms

$$n_i = N_i + S_i x\, F_i + n_{i+1} + P_i x\, f_{i+1} \qquad n_{n+1} = n_E \tag{45}$$

$$\tau_{ai} = n_i . z_i \tag{46}$$

2) Compute $r_0 = \tau - \tau_a$ (47)

Note that $G = 9.8061 m/s^2$ denotes the acceleration due to the gravity which is along the direction of z_1. The cost of this step is $n(87m+78a)-(21m+24a)$. The rest of the computation is carried out according to Eqs. (5)-(9) where the matrix-vector operation in Eq. (7) is performed by using the function $g_2(Q_a, p_j)$. Each iteration of Eqs. (5)-(9) requires $n(47m+46a)-(10m+23a)$ which, taking n iterations, leads to the total cost of the CCG algorithm as $n^2(47m+46a)+n(177m+118a)-(117m+87a)$. The cost of the developed algorithm is compared to the CCG algorithm of [7]. Note that the algorithm of this paper achieves a better efficiency by a significant reduction in the coefficient of n^2 terms.

IV. THE PCG-DS ALGORITHM

IV.1 An Algorithm for Computation of the Diagonal Elements of Inertia Matrix

From Eq. (3) the diagonal elements of the inertia matrix can be computed as

$$a_{ii} = \Gamma_i \tag{48}$$

for the conditions given by

$$\ddot{q}_i = 1 \text{ and } \ddot{q}_{k\neq i} = 0 \qquad \text{For } k = 1, 2, \ldots, n \tag{49}$$

An algorithm for computation of the terms a_{ii}, using g_3, can be derived as

$$a_{ii} = g_{3i}(Q_{ai}, e_i) \tag{50}$$

where subscript i denotes that g_3 is evaluated for the last n-i+1 links, Q_{ai} is the vector of actual position of the last n-i+1 joints. e_i is an ix1 vector as $e_i = [1\ 0\ \ldots\ 0]^t$. With the conditions given by Eq. (49), let $\dot{\omega}_{j(i)}$ and $\dot{V}_{j(i)}$, and $f_{j(i)}$ and $n_{j(i)}$ (j>i) stand for angular and linear acceleration of, and force and moment exerted on, link j (point O_j) due to the unit acceleration of joint i. For link j, Eqs. (28)-(31) are written as

$$\underline{\dot{\omega}}_{j(i)} = \underline{z}_i \tag{51}$$

$$\underline{\dot{V}}_{j(i)} = \underline{z}_i x \underline{P}_{j,i} \tag{52}$$

$$\underline{f}_{j(i)} = m_j(\underline{z}_i x \underline{P}_{j,i}) + \underline{z}_i x\, \underline{h}_j + \underline{f}_{j+1(i)} \tag{53}$$

$$\underline{n}_{j(i)} = \underline{\underline{k}}_j \underline{z}_i + \underline{h}_j x(\underline{z}_i x \underline{P}_{j,i}) + \underline{n}_{j+1(i)} + \underline{P}_j x \underline{f}_{j+1(i)} \tag{54}$$

and, for link i, these equation are written as

$$\underline{\dot{\omega}}_{i(i)} = \underline{z}_i \tag{55}$$

$$\underline{\dot{V}}_{i(i)} = 0 \tag{56}$$

$$\underline{f}_{i(i)} = \underline{z}_i x\, \underline{h}_i + \underline{f}_{i+1(i)} \tag{57}$$

$$\underline{n}_{i(i)} = \underline{\underline{k}}_i \underline{z}_i + \underline{n}_{i+1(i)} + \underline{P}_i x\, \underline{f}_{i+1(i)} \tag{58}$$

$$a_{ii} = \Gamma_i = \underline{z}_i \cdot \underline{n}_{i(i)} \tag{59}$$

Using Eqs. (51)-(54), Eqs. (57)-(58) can be rewritten as

$$\underline{f}_{i(i)} = \underline{z}_i x \underline{h}_i + \sum_{k=i+1}^{n} \left[m_k(\underline{z}_i x \underline{P}_{k,i}) + \underline{z}_i x \underline{h}_k \right] = \underline{z}_i x \left[\underline{h}_i + \sum_{k=i+1}^{n} \left[m_k \underline{P}_{k,i} + \underline{h}_k \right] \right]$$

$$= \underline{z}_i x \left[\underline{h}_i + \sum_{k=i+1}^{n} m_k \underline{P}_i + \sum_{k=i+1}^{n} \left[m_{k+1} \underline{P}_{k,i} + \underline{h}_k \right] \right] = \underline{z}_i x \underline{H}_i \tag{60}$$

$$\underline{n}_{i(i)} = \underline{\underline{k}}_i \underline{z}_i + \sum_{k=i+1}^{n} \left[\underline{\underline{k}}_k \underline{z}_i + \underline{h}_k x(\underline{z}_i x \underline{P}_{k,i}) + \underline{P}_{k,i} x [m_k(\underline{z}_i x \underline{P}_{k,i}) + \underline{z}_i x \underline{h}_k] \right]$$

$$= \left\{ \underline{\underline{k}}_i + \sum_{k=i+1}^{n} \left[\underline{\underline{k}}_k - m_k \hat{\underline{\underline{P}}}_{k,i} \hat{\underline{\underline{P}}}_{k,i} - \hat{\underline{h}}_k \hat{\underline{\underline{P}}}_{k,i} - \hat{\underline{\underline{P}}}_{k,i} \hat{\underline{h}}_k \right] \right\} \underline{z}_i = \underline{\underline{K}}_i \underline{z}_i \tag{61}$$

Note that the conditions given by Eq. (49) imply that link i through link n do not have any relative motion, i.e., are rigidly connected, and form a composite rigid-body. In comparison with Eqs. (57)-(58), $\underline{H}_i$ and $\underline{\underline{K}}_i$ represent the first and the second moment of mass of the composite system composed of link i through link n (denoted as composite system i) about point O_i. From Eqs. (60)-(61), the recursions for computation of $\underline{H}_i$ and $\underline{\underline{K}}_i$ are derived as

$$\underline{H}_i = \underline{h}_i + M_{i+1} \underline{P}_i + \left\{ \underline{h}_{i+1} + \sum_{k=i+2}^{n} \left[m_k \underline{P}_{k,i+2} + \underline{h}_k \right] \right. = \underline{h}_i + M_{i+1} \underline{P}_i + \underline{H}_{i+1} \tag{62}$$

where M_{i+1} is the mass (zeroth moment mass) of the composite system i+1, and

$$\underline{\underline{K}}_i = \underline{\underline{k}}_i + \sum_{k=i+1}^{n} \left[\underline{\underline{k}}_k - m_k \hat{\underline{\underline{P}}}_{k,i} \hat{\underline{\underline{P}}}_{k,i} - \hat{\underline{h}}_k \hat{\underline{\underline{P}}}_{k,i} - \hat{\underline{\underline{P}}}_{k,i} \hat{\underline{h}}_k \right] = \underline{\underline{k}}_i +$$

$$\sum_{k=i+1}^{n} \left[\underline{\underline{k}}_k - m_k (\hat{\underline{\underline{P}}}_{k,i+1} + \hat{\underline{\underline{P}}}_i)(\hat{\underline{\underline{P}}}_{k,i+1} + \hat{\underline{\underline{P}}}_i) - \hat{\underline{h}}_k (\hat{\underline{\underline{P}}}_{k,i+1} + \hat{\underline{\underline{P}}}_i) - (\hat{\underline{\underline{P}}}_{k,i+1} + \hat{\underline{\underline{P}}}_i) \hat{\underline{h}}_k \right]$$

which after some manipulations and by using Eq. (62) can be written as

$$\underline{\underline{K}}_i = \underline{\underline{k}}_i + M_{i+1} \hat{\underline{\underline{P}}}_i \hat{\underline{\underline{P}}}_i - \hat{\underline{\underline{H}}}_{i+1} \hat{\underline{\underline{P}}}_i - \hat{\underline{\underline{P}}}_i \underline{\underline{H}}_{i+1} + \left\{ \underline{\underline{k}}_{i+1} + \sum_{k=i+2}^{n} \left[\underline{\underline{k}}_k - m_k \hat{\underline{\underline{P}}}_{k,i+1} \hat{\underline{\underline{P}}}_{k,i+1} - \right. \right.$$

$$\hat{\underline{\underline{h}}}_k \hat{\underline{\underline{P}}}_{k,i+1} - \hat{\underline{\underline{P}}}_{k,i+1} \hat{\underline{\underline{h}}}_k \Big] = \underline{\underline{k}}_i - M_{i+1} \hat{\underline{\underline{P}}}_i \hat{\underline{\underline{P}}}_i - \hat{\underline{\underline{H}}}_{i+1} \hat{\underline{\underline{P}}}_i - \hat{\underline{\underline{P}}}_i \underline{\underline{H}}_{i+1} + \underline{\underline{K}}_{i+1} \qquad (63)$$

The diagonal elements of the inertia matrix, or M = Diag(A), are computed as

For i = n, n-1, ..., 1

$$M_i = M_{i+1} + m_i \qquad M_n = m_n \qquad (64)$$

$$\underline{H}_i = \underline{h}_i + M_{i+1} \underline{P}_i + \underline{H}_{i+1} \qquad \underline{H}_n = \underline{h}_n \qquad (65)$$

$$\underline{\underline{K}}_i = \underline{\underline{k}}_i + \underline{\underline{k}}_{i+1} - M_{i+1} \hat{\underline{\underline{P}}}_i \hat{\underline{\underline{P}}}_i - \hat{\underline{\underline{H}}}_{i+1} \hat{\underline{\underline{P}}}_i - \hat{\underline{\underline{P}}}_i \underline{\underline{H}}_{i+1} \qquad \underline{\underline{K}}_n = \underline{\underline{k}}_n \qquad (66)$$

$$a_{ii} = \underline{\underline{k}}_i \cdot \underline{z}_i \qquad (67)$$

It should be pointed out that Renaud [17] used the notion of augmented-body to derive the equations similar to Eqs. (62)-(63). However, our derivation of Eqs. (62)-(63) shows that this notion is implicit in the N-E formulation. The improved efficiency of the above algorithm, compared to the composite rigid-body algorithm in [7], results from the elimination of the redundancy in the intrinsic equations. Note that by directly computing the first and second moment of mass of composite system i about point O_i the redundant computation of the center of mass of composite systems and the force and moments acting on the centers of mass are avoided (see [8] for more discussion regarding these two algorithms).

IV.2 Computation of the PCG-DS Algorithm

Eqs. (64)-(67) describe the intrinsic relation between the diagonal elements of the inertia matrix, which are scalar, and the links kinematic and dynamics parameters, which consist of scalars, vectors, and tensors. In order to compute the diagonal elements, Eqs. (65)-(67) should be projected on some coordinate frame. Due to the evaluation of g_1 and g_2(or g_3) in EE frame, it is more efficient to project Eqs. (65)-(67) on this frame which allows the exploitation of synergism in different computations. To do so h_i and k_i need to be evaluated in EE frame and then H_i, K_i, and a_{ii} can be computed from Eqs. (65)-(67) with the vectors and tensors being described with respect to EE frame. The cost of evaluating M is then obtained as $(n-1)(51m+50a)$ where the symmetry of matrices in Eq. (66) is exploited. Note that the additional cost of PCG-DS algorithm, due to the evaluation of M, is almost equal to one iteration of Eqs. (5)-(9) for CCG algorithm. The best algorithm for computation of inertia matrix requires $n(69m+62a)-(57m+58a)$ for evaluation of the diagonal elements [8]. Hence, the geater efficiency of the developed algorithm results from the exploitation of synergism in different computation. Having computed M, the second step of the PCG-DS algorithm is performed similar to CCG algorithm and the rest of the computation is carried out according to Eqs. (10)-(15).

V. CONCLUSION

In this paper we investigated the applicability of conjugate gradient algorithms for computation of the manipulator forward dynamics. Two algorithms were presented and their computational efficiency was analyzed. The preconditioned algorithm is particularly promising because of its potentially rapid convergence as well as its suitability for parallel computation. We are currently investigating in greater detail the effect of

the preconditioner on the convergence of the algorithm. This work includes analysis of error estimates as well as simulations with actual manipulators.

ACKNOWLEDGEMENT

The research described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under the contract with the National Aeronautics and Space Administration (NASA).

References

1. M.H. Milman and G. Rodriguez, "Cooperative Dual Arm Manipulator Issues and Task Approach," JPL Eng. Memorandum (internal document), Nov. 1987.
2. J.Y.S. Luh, M.W. Walker, and R.P. Paul, "On-line Computation Scheme for Mechanical Manipulator," Trans. ASME J. Dyn. Syst., Meas., and Control, Vol. 102, pp. 69-76, June 1980.
3. R. Featherstone, *Robot Dynamics Algorithms*. Ph.D. Dissertation, Univ. of Edinburgh, 1984.
4. R. Featherstone, "The Calculation of Robot Dynamics Using Articulated-Body Inertia, "Int. J. Robotics Research, Vol.2(2), 1983.
5. G. Rodriguez, "Kalman Filtering, Smooting and Recursive Robot Arm Forward and Inverse Dynamics," IEEE Trans. Robotics&Automation, Vol. RA-5, Dec. 1987. Also in Jet Propulsion Laboratory Publication 86-48, Dec. 1986.
6. G. Rodriguez and K.K. Kreutz, "Recursive Mass Matrix Factorization and Inversion: An Operator Approach to Open and Closed Chain Multibody Dynamics," Jet Propulsion Laboratory Publication 88-11, May 1988.
7. M.W. Walker and D.E. Orin, "Efficient Dynamic Computer Simulation of Robotics Mechanisms," Trans. ASME J. Dyn. Syst., Meas., Control, vol. 104, pp. 205-211, Sept. 1982.
8. A. Fijany and A.K. Bejczy, "An Efficient Method for Computation of the Manipulator Inertia Matrix," Submitted to the J. of Robotic systems.
9. A. Fijany, *Parallel Algorithms and Architectures in Robotics*. Ph.D. Dissertation, Univ. of Paris XI (Orsay, Paris Sud), Sept. 1988.
10. A. Fijany and A.K. Bejczy, "Parallel Algorithms and Architecture for Computation of the Manipulator Forward Dynamics," Submitted to IEEE Trans. Syst., Man, and Cybernetics.
11. A. Fijany and A.K. Bejczy, "Parallel Algorithms for Computation of the Manipulator Inertia Matrix," Submitted to IEEE J Robotics & Automation.
12. G.H. Golub and C.F. Van Loan, *Matrix Computation*. The Johns Hopkins Univ. Press, Baltimore, Maryland, 1983.
13. D.P. O'Leary, "The Block Conjugate Algorithm and Related Methods," Linear Algebra and its Applications, Vol. 29, pp. 293-322, 1980.
14. M. K. Seager, "Parallelizing Conjugate Gradient for the Cray X-MP," Parallel Computing, Vol. 3, pp. 35-47, 1896.
15. P. Concus, G.H.Golub, and D.P. O'Leary, "A Generalized Conjugate Gradient Method for the Numerical Solution of Elliptic Partial Differential Equations," in *Sparse Matrix Computation*, J. R. Bunch, and D.J. Rose (Eds.), Academic Press, New York, 1976.
16. J. Chen, "The Effects of Gear Reduction on Robot Dynamics," Proc. of NASA Conf. on Space Telerobotics, JPL Publication 89-7 (this proceedings).
17. M. Renaud, "An Efficient Iterative Analytical Procedure for Obtaining a Robot Manipulator Dynamic Model," Proc. 1st. Int. Symp. on Robotics Research, 1983.
18. C.S.G. Lee and P.R. Chang, "Efficient Parallel Algorithms for Robot Forward Dynamics Computation," IEEE Trans. Syst., Man, and Cybern., Vol. 18(2), pp. 238-251, March/April 1988.